MCCORMICK, PAULDING & HUBER LLP
City Place II
185 Asylum Street
Hartford, CT 06103-4102
Tel. (860) 549-5290

"EXPRESS MAIL" MAILING LABEL
NUMBER EV337023864US
DATE OF DEPOSIT December 4, 2003
I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 C.F.R. 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, MAIL STOP PATENT APPLICATION, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450

LISA M. TINNIRELLA
(TYPED NAME OF PERSON MAILING PAPER OR FEE)
Lisa M. Tinnirella
(SIGNATURE OF PERSON MAILING PAPER OR FEE)

Mail Stop **PATENT APPLICATION**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

ATTACHMENT TO A PATENT APPLICATION

DOCKET NO.:	5227-01-1
ENTITLED:	DATA PROCESSING SYSTEM HAVING A CARTESIAN CONTROLLER
INVENTOR(S):	DAN TOMESCU and GHEORGHE STEFAN
INCLUDING:	Specification; Claims; Abstract; and two (2) sheets of Informal Drawings

DATA PROCESSING SYSTEM HAVING A CARTESIAN CONTROLLER

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/431,154 entitled "ENHANCED VERSION OF CONNEX MEMORY", and filed on December 5, 2002, hereby incorporated by reference in its entirety. The subject matter of this application relates to pending U.S. Patent Application Serial No. 09/928,151 entitled "A MEMORY ENGINE FOR THE INSPECTION AND MANIPULATION OF DATA", filed on August 10, 2001 and U.S. Patent Application Serial No. ________________ entitled "CELLULAR ENGINE FOR A DATA PROCESSING SYSTEM", filed December ______, 2003, both of which are herein incorporated by reference in their entirety.

FIELD OF THE INVENTION

[0002] The invention relates generally to a data processing system having a controller, and more particularly, to a data processing system having a controller which acts as a stack-oriented processor and, in one embodiment, is capable of driving the execution of the Connex Engine (CE) and thus the operation of the Connex Memory (CM) contained therein.

BACKGROUND OF THE INVENTION

[0003] Controllers are utilized in many, if not all, complex computerized systems. Typically, these controllers take the form of known processors, or the like, for controlling the sequential operation of those components or devices which are in communication with the controller.

[0004] In such systems, it is oftentimes necessary for large amounts of data to be processed, searched or otherwise manipulated in short timeframes. Typically, the speed at which these operations are executed can be increased by configuring the overall system such that the controlled components each operate in parallel with one another. That is, instructions from the controller in such systems are carried out by the various components of the system in parallel every cycle, thus saving time and increasing the efficiency of the system.

[0005] There exists, however, certain inherent inefficiencies in the operation of known processor-controlled systems. Perhaps most significantly, known processor-controlled systems exhibit significant 'down-time' when required to not only issue instructions to those components connected in parallel, but also to execute basic routines within the controller itself. That is, known processor-controlled systems are incapable of executing operations within themselves while also seeing to the parallel execution of instructions in their integrated components, all within a common time-frame.

[0006] With the forgoing problems and concerns in mind, the present invention therefore seeks to utilize a data processing system having a new type of controller termed a Cartesian Controller (CC) which is capable of issuing instructions for the parallel execution of these instructions in controlled devices, as well as performing operations within itself, all within a single clock cycle.

SUMMARY OF THE INVENTION

[0007] It is an object of the present invention to provide an efficient data processing system.

[0008] It is another object of the present invention to provide an efficient data processing system which is capable of issuing instructions to a plurality of controlled devices.

[0009] It is another object of the present invention to provide an efficient data processing system which is capable of ensuring that instructions are executed within a plurality of controlled devices in parallel with one another.

[0010] It is another object of the present invention to provide an efficient data processing system which is capable of performing internal operations as well as issuing instructions to a plurality of controlled devices.

[0011] It is another object of the present invention to provide a controller for an efficient data processing system which is capable of performing internal operations as well as issuing instructions to a plurality of controlled devices, whereby the internal operations of the controller and the parallel processing in the controlled devices are both accomplished within every clock cycle.

[0012] According to one embodiment of the present invention, a data-processing system for the selective manipulation of data includes a cellular data processing engine A controller is utilized which selectively outputs one of a plurality of instructions to the cellular engine for driving the execution of the programs enabled by the engine, while a clock device is utilized for outputting a synchronizing clock signal comprised of a predetermined number of clock cycles per second. The clock device outputs the synchronizing clock signal to the cellular engine and the controller. The controller outputs one of the instructions to the cellular engine for execution of one of the programs, while also executing an operation within itself, all within a single clock cycle.

[0013] These and other objectives of the present invention, and their preferred embodiments, shall become clear by consideration of the specification, claims and drawings taken as a whole.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Figure 1 is a block diagram showing the general architecture of a Cartesian Controller, including a memory engine and a synchronizing clock element, according to one embodiment of the present invention.

[0015] Figure 2 exhibits the instruction format for the Cartesian Controller.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0016] Figure 1 depicts the architectural relationship between a Cartesian Controller (CC) **100** as it relates to a data processing engine **102**, hereinafter referred to as the Connex Engine (CE), and an array of active cells, or processing elements, **104**, hereinafter referred to as the Connex Memory (CM), supported therein. A synchronizing clock circuit **106** is utilized to coordinate the operation of the CC **100**, the CE **102** and the CM **104** such that one of a plurality of instructions may be issued by the CC **100** and transferred to the CE **102** and the CM **104** for parallel execution and processing.

[0017] The clock circuit **106** is capable of outputting a predetermined number of clock cycles per second, and the CC **100** is capable of performing an internal operation such that the CC **100** performs one of a plurality of internal operations while also issuing one of a plurality of instructions to the CE **102** within a single clock cycle. A buffer memory **108** and a RAM controller **110** are also shown.

[0018] The buffer memory **108** is organized as a pool of buffers, each preferably having a size equal to the size of the CM **104**, and which is under the control of the CE **102**. The purpose of these buffers, also referred to as lines, is to allow for search, insert and delete operations, amongst other operations, to be performed on character strings longer than may be accommodated within the CM **104**, and thus offering a lower cost of implementation and a reduction in power dissipation.

[0019] It will be readily appreciated that the present invention contemplates that the CC **100** may have any number of circuit-specific configurations without departing from the broader aspects of the present invention provided that the CC **100** is capable of issuing commands to, and receiving data from, the CE **102**.

[0020] In accordance with the preferred embodiment of the present invention, the CC **100** is a 32-bit stack-oriented processor which is capable of driving the execution of programs enabled in the CE **102**. The CE **102** is itself a data-parallel machine, also known as a SIMD (Single Instruction Multiple Data) machine. The code that the CE **102** executes is sequential, but the operation specified by each instruction that the CE **102** gets from the CC **100** is executed in parallel by all cells in the CM **104** within one clock cycle. That is, the code of an operation like "find" is fed into a plurality of available cells that all do their work strictly in parallel, independently of each other.

[0021] While the present invention has been described as enabling the parallel execution within the CE **102**, the present invention is not so limited in this regard as other machines and devices may be alternatively attached so as to be in communication with the CC **100**, without departing from the broader aspects of the present invention. That is, the present invention envisions that other devices apart from, or instead of, the CE **102**, may be controlled in parallel, stemming from instructions issued by the CC **100**, all within a single clock cycle.

[0022] Returning to Figure 1, the CE **102** does not itself have access to a program memory to fetch its instructions – every cycle its cells expect to get an operation code in a special register, but it takes a different entity, in the present case, the CC **100**, to do it for them; the code is sequential and there needs to be a single point of access to fetch it. The main job of the CC **100** is therefore to drive the execution of CE **102** programs, i.e. fetch instructions to be executed by individual cells and place them in an internal register; at the same time, it serves as a gateway to CE **102** and thereby takes care of all input/output interactions. The CC **100** also executes simple sequential operations without which it would be impossible to write meaningful code for such a machine: one class of such operations are the so-called "control primitives", i.e. those instructions that are used to code decision-making sequences (e.g. *if*, *while*, *repeat*, etc).

[0023] While a sequential controller is typically utilized to drive the execution of code for data-parallel machines, known architectures utilize one of the two following solutions in the implementation of their respective controllers:

1. A conventional processor, such as an off-the-shelf chip, like an ARM® controller or an Intel® processor. The efficiency of this solution is not optimal because when these controllers execute their own instructions, the parallel machine(s) to which they are connected are idle. That is, known conventional processors are not capable of fetching instructions for their connected parallel machines, while also executing their own instructions; and

2. A microprogrammed machine using a plurality of microinstruction fields to issue commands to a set of devices. The efficiency of this solution is not optimal because of the difficulty of coordinating the parallel execution of two independent instructions belonging to different instruction sets that are both implemented in microcode.

[0024] It is therefore an important aspect of the present invention that the CC **100**, which was designed and optimized with the CE **102** and the CM **104** in mind but can also serve to control other machines, implements an instruction pipeline that allows for the parallel execution of two types of instructions per clock cycle: one for the CE **102**, and another for the CC **100** itself.

[0025] As will be appreciated, by enabling the execution of two types of instructions within each clock cycle, the CC **100** is capable of performing data handling operations in a much more efficient and time sensitive manner. In particular, the CC **100** is capable of controlling the execution of code belonging to two different instruction sets in a single clock cycle, whereas known sequential controllers and their parallel machines take two, or more, clock cycles to accomplish the same tasks.

[0026] The CC **100** has a RISC architecture and, as mentioned previously, is capable of performing an internal operation while also generating a corresponding operation within the CE **102**, all within a single clock cycle. The instruction for both of the operations in the CC **100** and the CE **102** is thus encoded in each instruction of the CC **100**.

[0027] In a preferred embodiment of the present invention, the following resources are part of the structure of the CC **100**, which is a stack-based machine:

1. A control unit, whose main components are a program memory and a control stack;

2. An execution unit, whose main components are a data memory, a data stack and a logic and arithmetic unit;

3. An I/O unit, whose main components are two FIFOs; and

4. A co-machine interface, containing mainly drivers and receivers.

Instruction Set for the CC 100

[0028] Figure 2 illustrates the format of CC **100** instructions, in which the system performs, in parallel, an instruction in the CE **102**, as encoded in the *ceOp* field, as well as an instruction in the CC **100**, as encoded in the *ccOp* field.

[0029] Instructions include the following fields: *ccOp*, located in instr[31:25], is an operation code for the CC **100**; *ceOp*, located in instr[24:17], is an operation code for the CE **102** (sent to the CE **102** and executed in parallel with the instruction encoded in the ccOp field); *value*, located in instr[16:0], is an immediate operand for either instruction, which may be utilized for both instructions, if desired.

[0030] Instructions implemented by the preferred embodiment of the CC **100** fall into the following categories:

1. Arithmetic and logic instructions;

2. Data transfer instructions;

3. Stack instructions, including:

a. **remove** <*num*>, which pops a number of values equal to the *num* operand off the top of the data stack by subtracting the value of *num* from the data stack pointer;

b. **restore**, which restores the value of the stack pointer to what it was before the last execution of the **remove** instruction;

c. **pushframe** <*num*>, which pushes a number of words equal to the *num* operand onto the control stack by adding the *num* value to the control stack pointer. Note that the contents of the newly pushed control stack frame are undefined; and

d. **popframe** <*num*>, which pops a number of words equal to the *num* operand off the top of the control stack by subtracting the *num* value from the control stack pointer.

4. Stack interchange instructions, including:

a. **xchg**, which which swaps the top values of the data stack and control stack respectively;

b. **cs2ds**, which pops a value off the top of the control stack and pushes it onto the data stack;

c. **ds2cs**, which pops a value off the top of the data stack and pushes it onto the control stack; and

d. **pushcs** <*num*>, which pushes onto the data stack a control stack value whose offset relative to the control stack pointer is given by the *num* operand.

5. Control instructions;

6. Wait instructions, of the form '**w** *event*', which take one or more cycles to complete. Within each cycle of a wait-type instruction a certain Boolean condition called *event* is evaluated; if the truth value of the *event* is true, then the CC program counter is incremented; otherwise the program counter remains unchanged, the consequence being that the same condition will also be tested

within the following clock cycle. Some of the *events* associated with wait-type instructions in the preferred embodiment of the CC **100** are:

a. *zero*: the top of the data stack is compared to zero and if not equal to zero it is decremented, otherwise the top data stack value is removed;

b. *noMarker*: a flag set by the control system, i.e., the CE **102** in the preferred embodiment of the CC **100**;

c. *noFullOut*: a flag which, if true, indicates that the output FIFO is ready to accept new data; and

d. *noEmptyIn*: a flag which, if true, indicates that the input FIFO contains at least one word of data.

7. Test instructions;

8. Conditional jump instructions; and

9. I/O instructions.

[0031] Although the preceding list offers instructions implemented by the preferred embodiment of the CC **100** in specified categories, it will be readily appreciated that the CC **100** may implement other, additional instructions in non-specified categories without departing from the broader aspects of the present invention. Indeed the preceding list is offered by way of explanation and example and should not be construed as exhaustive even within the categories listed above.

[0032] While the invention had been described with reference to the preferred embodiments, it will be understood by those skilled in the art that various obvious changes may be made, and equivalents may be substituted for elements thereof, without departing from the essential scope of the present invention. Therefore, it is intended that the invention not be limited to the particular embodiments disclosed, but that the invention includes all embodiments falling within the scope of the appended claims.